User Manual

Programming components - PUE5

Manual number:
ITKU-48-01-11-09-PL

MANUFACTURER OF ELECTRONIC WEIGHING INSTRUMENTS

RADWAG 26 – 600 Radom Bracka 28 Street - POLAND
Phone +48 48 38 48 800, phone/fax. +48 48 385 00 10
Selling department +48 48 366 80 06
www.radwag.com

SPIS TREŚCI

1. BIBLIOTEKA DLL DO KOMUNIKACJI Z MODUŁEM MW-02 5
1.1. Intended Use 5
1.2. Installation 5
1.3. Functions 6
1.3.1 Communication with a scale 6
1.3.2 Operations on the measurement_st structure 9
1.3.3 Scale tarring and zeroing 16
1.3.4 Operations on parameters 18
1.3.5 Analogue output 21
1.3.6 Operations on I/O 21
2. ACTIVEX CONTROL FOR MW-02 MODULES 26
2.1. Intended use 26
2.2. Installation 26
2.3. Configuration 29
2.3.1 General parameters 29
2.3.2 Colors 30
2.3.3 Units 31
2.3.4 Events 31
2.4. Operating 32
2.4.1 Button's functions 33
2.4.2 Error messages 33
2.5. Accessible Methods 34
2.6. Generated events 35
3. DYNAMIC-LINK LIBARIES (DLL) FOR PUE C/31 36
3.1. Intended Use 36
3.2. Installation 36
3.3. Functions 36
3.3.1 Communication with a scale 36
3.3.2 Operations on structure measurement_st 41
3.3.3 Reading weighing unit in string format 43
3.3.4 Tarring and zeroing 47
3.3.5 Operations on parameters 49
3.4. PueC31 controller configuration 51
4. ACTIVEX CONTROL FOR INDICATORS PUE C/31 52
4.1. Intended Use 52
4.2. Installation 52
4.3. Configuration 55
4.3.1 General parameters 55
4.3.2 Colors 56
4.3.3 Events 57
4.4. Operating 58
4.4.1 Functions of buttons 58

4.4.2 Error messages 59
4.5. Accessible methods 59
4.6. Generated events 60
4.7. PueC31 controller configuration 60
5. PROFIBUS COMMUNICATION MODULE 62
5.1. Intended use 62
5.2. Data exchange 62
5.3. Diagnostics and operations on the ProfiBus module 63
5.4. Memory map 64
5.5. Configuration 64
5.6. Cable for module PROFIBUS 64
5.7. All fields used in Profibus communication 65
5.8. PROFIBUS configuration example 66
5.9. Content of CD 67
6. COMPONENT OF COMMUNICATION WITH BAR-CODE SCANNER. 68
6.1. Intended use 68
6.2. Installation 68
6.3. Properties 68
6.3.1 Form - Ctrl 68
6.3.2 Serial port number – PortNumber 68
6.3.3 Baud rate – PortSpeed 68
6.3.4 Telegram prefix length – PrefixNumber 68
6.3.5 Telegram suffix length – PostfixNumber 68
6.4. Methods 69
6.4.1 Starting reception telegrams from a scanner 69
6.4.2 Completion of reception of telegrams from a scanner 69
6.5. Events 69
6.5.1 Reading a new code 69

1. BIBLIOTEKA DLL DO KOMUNIKACJI Z MODUŁEM MW-02

1.1. Intended Use

The DLL's for MW-02 weighing modules in PUE 5 weighing terminals are intended for external applications. They make accessible all functionalities of the weighing modules for computer programmers and allow them to create their own weighing software or use only some features of the module.

1.2. Installation

Package files:

- **PUE5client.dll**
  This .dll file should be placed in the folder where the executable file of the created program is placed,
- **PUE5client.lib**
  This file is intended to attach to customers' C++ projects,
- **Pue5Client.h**
  The library header, intended to attach to customers' C++ projects,
- **PUE5client.cs**
  Libraries implemented in the class, intended to attach to customers' C# projects.

Notice:
*The functions included in the library cooperate as client-server "**PUE 5 scale driver**" so it is essential to have it installed on the weighing terminal.*

1.3. Functions

1.3.1 Communication with a scale

In order to communicate with the scale the following parameters should be set:

- **IP** – weighing terminal address with installed the "**PUE 5 scale driver**",
- **address** – the **MW-02** module address. In standard installations it is possible to have two modules with addresses set to 1 and 2, for the module I / O set address to 254.
- **timeout** – the maximal time interval for receiving a response from a module,
- **No_repeat** – the number of failure tries in establishing communication with a module that causes the error message.

All this settings are put into the ***com_conf_st*** structure using the ***set_com_conf*** function.

Caution:

*It should be remembered that the memory taken by the **com_conf_st** structure, should be released using the **free_com_conf_st** function.*

1.3.1.1 Creating the structure com_conf_st structure comprising communication parameters.

com_conf_st set_com_conf(char* _IP, int _address, int _timeout, int _repeat_number);*

Parameters:

- ***IP*** – scale IP address,
- ***address*** – device address connected to the scale,
- ***timeout*** - the maximal time interval in milliseconds for receiving a response from an external device,
- ***repeat_number*** - the number tries to establish communication with a device.

Returned value:

Pointer to the structure comprising communication parameters.

1.3.1.2 Function releasing memory occupied by the com_conf_st structure

int free_com_conf_st(com_conf_st config);*

Parametrers:

- Pointer to the structure ***com_conf_st*** comprising communication parameters.

Returned value:

1 – memory released

1.3.1.3 Sending request and receiving responses from internal RS devices

- *int request_com_conf(com_conf_st* config, const char* query_text, char* response_buffer);*

Parameters:

- ***config*** – pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,
- ***query_text*** – string of characters comprising a request terminated by '\0',
- ***response_buffer*** – buffer to which the function inscribes a response, allocated array of characters.

Returned value – Operation status:

- 0 – operation successful, status OK
-1 – device timeout
-2 – CRC error
-3 – response string to short
-4 – Frame beginning inappropreate
-5 – port cannot be opened
-6 – another exception (server)
-7 – communication timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – another error (client)

In case of points -6,-9 – details about an excerption/error can be found

- *int request(const char* IP, int adress, const char* query_text, int timeout, char* response_buffer, int repeat_number);*

Parameters:

- ***IP*** – scale IP address,
- ***address*** - device address connected to the scale,
- ***query_text*** - string of characters comprising a request terminated by '\0',
- ***timeout*** - the maximal time interval in milliseconds for receiving a response from an external device,
- ***response_buffer*** - buffer to which the function inscribes a response, allocated array of characters,
- ***repeat_number*** - the number tries to establish communication with a device.

Returned value – Operation status:

- 0 – operation successful, status OK
-1 – device timeout
-2 – CRC error
-3 – response string to short
-4 – Frame beginning inappropriate
-5 – port cannot be opened
-6 – another exception (server)
-7 – communication timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – another error (client)

In case of points -6,-9 – details about an excerption/error can be found

1.3.1.4 Function releasing memory occupied by com_conf_st structure

int free_com_conf_st(com_conf_st config);*

Parameters:

- ***config*** – pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,

Returned value:

1 – released memory

1.3.2 Operations on the measurement_st structure

The full measurement information comprises:

- Mass value,
- Tare value,
- Weighing unit code,
- Decimal point position,
- Scale status.

All these data are included in the ***measurement_st*** structure and can be read by using the function ***get_measurement_st***. The structure fields can be read by dedicated functions.

Caution:

*It should be remembered that memory occupied by the **measurement_st** structure needs to be released using the function **free_measurement_st**.*

1.3.2.1 Scale measurement reading

measurement_st get_measurement_st(com_conf_st* config);*

Parameters:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,

Returned value:

Pointer to the structure ***meas*** holding the measurement.

1.3.2.2 Releasing memory occupied by the measurement_st structure

int free_measurement_st(measurement_st measurement);*

Parameters:

- ***measurement***t - pointer to the structure ***meas*** holding the measurement.

Returned value:

1 – memory released

1.3.2.3 Reading mass value in floating point format

float get_weight_float(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

Read value.

1.3.2.4 Reading mass value in string format

int get_weight_string(measurement_st measurement, char* weight);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement,
- ***weight*** – pointer to the allocated array of characters for mass value.

Returned value:

Reading operation status.

1.3.2.5 Checking stability of the scale

int is_status_stable(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value – stability status:

1 - stable
0 – non-stable

1.3.2.6 Reading weighing unit in string format

int get_unit_string(measurement_st measurement, char* unit);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.
- ***unit*** – pointer to the allocated array of characters for unit value.

Returned value:

Reading operation status.

1.3.2.7 Reading mass value in fixed point format

int get_unit_int(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

Measuring unit.

```
enum units
{
    g = 0,
    kg = 1,
    lb = 2,
};
```

1.3.2.8 Reading tare value in floating point format

float get_tare_float(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

Tare value.

1.3.2.9 Reading tare value in string format

int get_tare_string(measurement_st measurement, char* tare);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement,
- ***tare*** – pointer to the allocated array for tare value.

Returned value:

Reading operation status.

1.3.2.10 Reading of decimal point position

int get_fraction_lenght(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

Decimal point position.

1.3.2.11 Reading device operation mode

modes get_mode(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

Operation mode:

```
enum modes
{
    Start_zeroing = 0, // Initial zeroing
    Zeroing = 1, // Zeroing
    Tarring = 2, // Tarring
    Start_mass_setting = 3, // start mass adjustment
    Calib_coeff_calc = 4, // span adjustment
    Static_weighting = 5 // static weighing
};
```

1.3.2.12 Checking if a measurement is correct

int is_correct_weight(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

1 – correct measurement
0 – faulty measurement

1.3.2.13 Checking if the scale has been zeroed

int is_status_zero(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value - zero status:

1 – zeroed
0 – out of zero

1.3.2.14 Checking if the scale has been tarred

int is_status_tare(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value – tare status:

1 – Tare > 0
0 – Tare = 0

1.3.2.15 Checking if the range is exceeded

int is_status_FULL(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value - FULL status:

1 – over the range
0 – within the range

1.3.2.16 Checking if a measurement is below the range

int is_status_NULL(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value - NULL status:

1 – below the range
0 – within the range

1.3.2.17 Checking the LH error – start mass behind the allowed range

int is_status_LH(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value – LH status :

1 – start mass behind the acceptable range (LH error)
0 - start mass within the acceptable range

1.3.2.18 Checking if weighing is within the second range

int is_status_range2(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value - range2 status :

1 – mass within the second range (over the first range)
0 – mass below or over the second range

1.3.2.19 Checking if weighing is within the third range

int is_status_range3(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value – range3 status :

1 – mass within the third range (over the second range)
0 – mass below or over the third range

1.3.2.20 Checking status of long lasting operations such as zeroing or tarring

int get_long_operation_status(measurement_st measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value – long lasting operation status:

-5 – operation time error
-6 – tarring behind the allowed range
-7 – zeroing behind the allowed range
0 – status ok

1.3.3 Scale tarring and zeroing

1.3.3.1 Setting tare value

int set_tare(com_conf_st config, float _tare);*

Parameters:

- ***config*** – pointer to the structure ***com_conf_st*** comprising communication parameters,
- ***tare*** – tare value to be set.

Returned value – operation status:

0 - Done, status OK

-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer response_buffer not allocated
-9 – other error (client)
-102 – wrong parameter value
-103 – operation cannot be executed

1.3.3.2 Scale tarring

int tarring(com_conf_st config);*

Parameter:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)

1.3.3.3 Scale zeroing

int zeroing(com_conf_st config);*

Parameter:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout

-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)
-103 – operation cannot be executed

1.3.3.4 Reading status of tarring and zeroing

int long_operation_status(com_conf_st config);*

Parameters:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,
- ***status*** – Pointer to the variable of ***int*** type for long operation status code.

Returned value – operation status:

-5 – operation time error
-6 – tarring behind the allowed range
-7 – zeroing behind the allowed range
-8 – scale communication error
0 – status ok

1.3.4 Operations on parameters

Both reading and setting parameters requires two steps:

- Copying parameters to the RAM (Random Access Memory),
- Reading a parameter from the RAM,
- Changing a parameter in the RAM,
- Saving a parameter from the RAM to non-volatile module memory.

1.3.4.1 Reading parameters from non-volatile module memory to RAM

int read_parameters(com_conf_st config);*

Parameter:

- ***config*** - pointer to the structure ***com_conf_st*** comprising communication parameters.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)

1.3.4.2 Reading parameters form RAM

int get_parameter(com_conf_st config, int number, char* buf);*

Parameters:

- ***config*** - pointer to the structure ***com_conf_st*** comprising communication parameters,
- ***number*** – parameter number,
- ***buf*** – allocated array of characters for parameters' values.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)

1.3.4.3 Changing a parameter value in the RAM

int set_parameter(com_conf_st config, int number, char* value);*

Parameter:

- ***config*** - pointer to the structure ***com_conf_st*** comprising communication parameters,
- ***number*** – parameter number,
- ***value*** – parameter value.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)

1.3.4.4 Saving parameters from RAM in non-volatile memory

int write_parameters(com_conf_st config);*

Parameter:

- *config* - pointer to the structure ***com_conf_st*** comprising communication parameters.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer *response_buffer* not allocated
-9 – other error (client)

1.3.5 Analogue output

int set_analog_output(com_conf_st config, float percent)*

Parameters:

- *config* - pointer to the structure ***com_conf_st*** comprising communication parameters
- *percent* – output value in per cents of the output range.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer *response_buffer* not allocated
-9 – other error (client)

1.3.6 Operations on I/O

1.3.6.1 Read inputs

int get_inputs(inputs_st* inputs, com_conf_st* config)

Parameters:

- inputs – pointer to an allocated array with inputs' addresses (masks),
- config – pointer to structure com_conf_st comprising communication parameters.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout

-8 - Buffer not allocated
-9 – other error (client)
-102 – wrong parameter value
-103 – operation cannot be executed

1.3.6.2 Function comprising addresses (masks) of inputs

int get_input(int number, inputs_st* inputs)

Parameter:

- number – input number, for which we want to get the mask,
- inputs – pointer to the array with input ports masks.

Returned value – operation status:

-1 – input number from behind the scope

1.3.6.3 Function checking the previous reading state for OR function

int get_input_or(int number, inputs_st* inputs)

Parameters:

- number – output number to check the state of the previous readout,
- inputs – pointer to the array with input ports masks.

Returned value – operation status:

-1 – input number from behind the scope

1.3.6.4 Function checking the previous reading state for AND function

int get_input_and(int number, inputs_st* inputs)

Parameters:

- number – output number to check the state of the previous readout,
- inputs – pointer to the array with input ports masks.

Returned value – operation status:

-1 – input number from behind the scope

1.3.6.5 Function for reading outputs

int get_outputs(int* state, com_conf_st* config)

Parameters:

- state – array alocated to outputs' addresses
- config – pointer to structure com_conf_st comprising communication parameters.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer not allocated
-9 – other error (client)
-102 – wrong parameter value
-103 – operation cannot be executed

1.3.6.6 Function setting outputs

int set_outputs(int mask, int state, com_conf_st* config)

Parameters:

- mask – output mask to be set,
- state – parameter setting output e.g. on or off.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer not allocated
-9 – other error (client)
-102 – wrong parameter value
-103 – operation cannot be executed

1.3.6.7 Function setting output

int set_output(int number,com_conf_st* config)

Parameter:

- number – output number,
- config – pointer to structure com_conf_st comprising communication parameters.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer not allocated
-9 – other error (client)
-102 – wrong parameter value
-103 – operation cannot be executed

1.3.6.8 Function disabling output port

int clear_output(int number,com_conf_st* config)

Parameters:

- number – output number to switch off,
- config – pointer to structure com_conf_st comprising communication parameters.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-2 – CRC error
-3 – to short response
-4 – Wrong frame beginning
-5 – Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer not allocated
-9 – other error (client)
-102 – wrong parameter value
-103 – operation cannot be executed

2. ACTIVEX CONTROL FOR MW-02 MODULES

2.1. Intended use

ActiveX control for **MW-02** modules of **PUE 5** weighing terminal, is intended to create scale interfaces on the terminal screen.

This control can be used in programs written in programming environments and languages that support **ActiveX** technology.

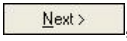
Caution:
*This control works as **client-server** with **„PUE 5 controller”** installed on the weighing terminal. You can install the controller from the CD that is attached to the terminal.*

2.2. Installation

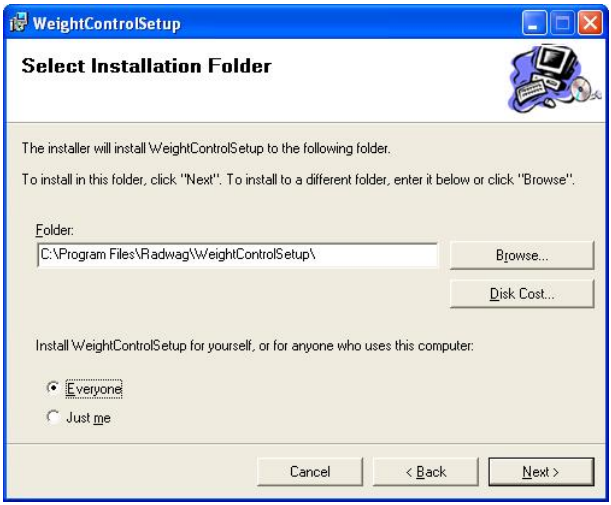
Installation procedure for ActiveX control:

- Run: **setup.exe**, the following window is displayed:

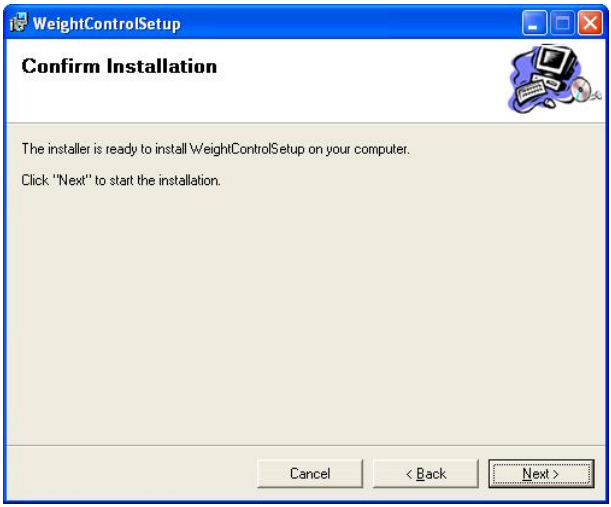
WeightControlSetup
Welcome to the WeightControlSetup Setup Wizard
The installer will guide you through the steps required to install WeightControlSetup on your computer.
WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.
Cancel
< Back
Next >

- Go to the next step by pressing 

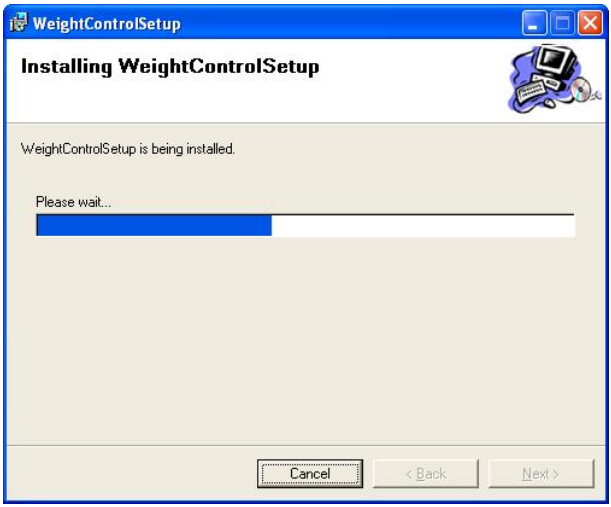
,
- Program **Setup.exe** allows the user to chose the folder name for the program:


WeightControlSetup
Select Installation Folder
The installer will install WeightControlSetup to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files\Radwag\WeightControlSetup\
Browse...
Disk Cost...
Install WeightControlSetup for yourself, or for anyone who uses this computer:
Everyone
Just me
Cancel
< Back
Next >

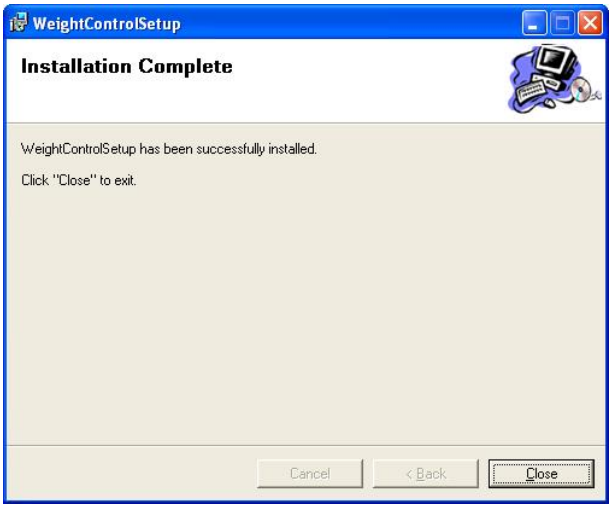
- Go to the next step by pressing Next >,
- Program **Setup.exe** informs the user that it is ready to be installed:


WeightControlSetup
Confirm Installation
The installer is ready to install WeightControlSetup on your computer.
Click "Next" to start the installation.
Cancel
< Back
Next >

- Go to the next step by pressing Next > and program **Setup.exe** begins the installation process:



- When the installation is completed the following window is displayed:



- Press Close and the installation process is completed.

Caution:

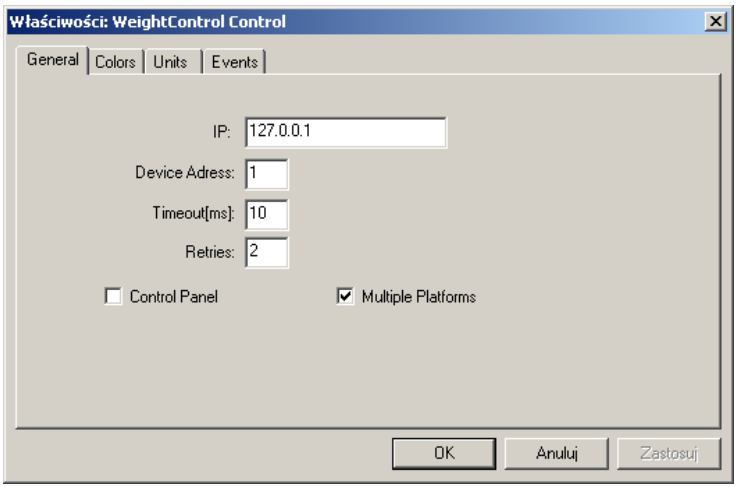
1. *The installation of **„PUE 5 controller”**, as well as the installation of ActiveX control, is simple, intuitive and do not require additional remarks. If there are no special requirements just accept the default way of installation.*
2. *For appropriate operation of **ActiveX** control it is necessary to install both the **„PUE 5 controller”** and the control on the terminal.*

2.3. Configuration

The look and operation of the control are strongly parameterized. Default values allow to run the control on the weighing terminal. Parameters are divided into four subsets.

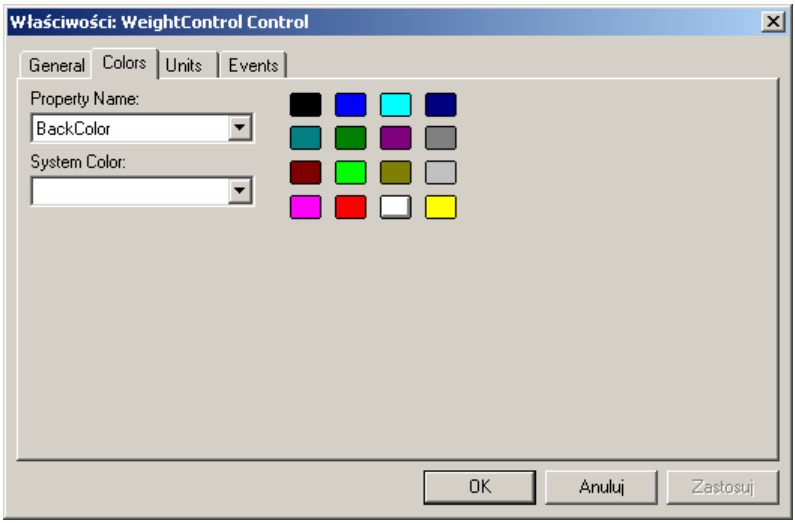
- “General” parameters include setting of internal communication with the weighing module and enabling/disabling the appearance of buttons.
- “Colours” enables you to change the appearance of the control.
- “Units” allow you to enable different weighing units and set the main unit.
- “Events” outlines the conditions that generate program different events .

2.3.1 General parameters



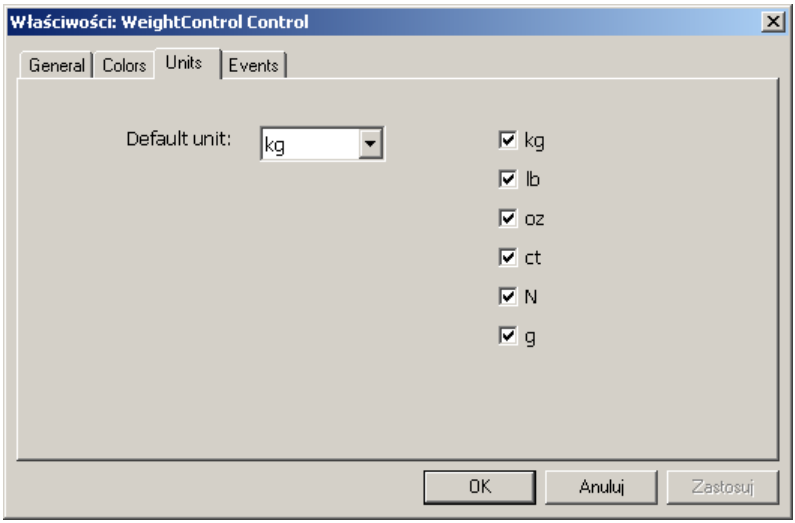
- IP – Internet Protocol Address of weighing terminal.
- Device Address – Weighing module address (set by the slot the module is in).
- Timeout – maximal time measured in [ms] the terminal waits for any module response.
- Retries – Number of failure module readings that generate an error.
- Control Panel – if ticked buttons are visible.
- Multiple Platforms – if ticked and buttons are visible then platform change button is visible.

2.3.2 Colors



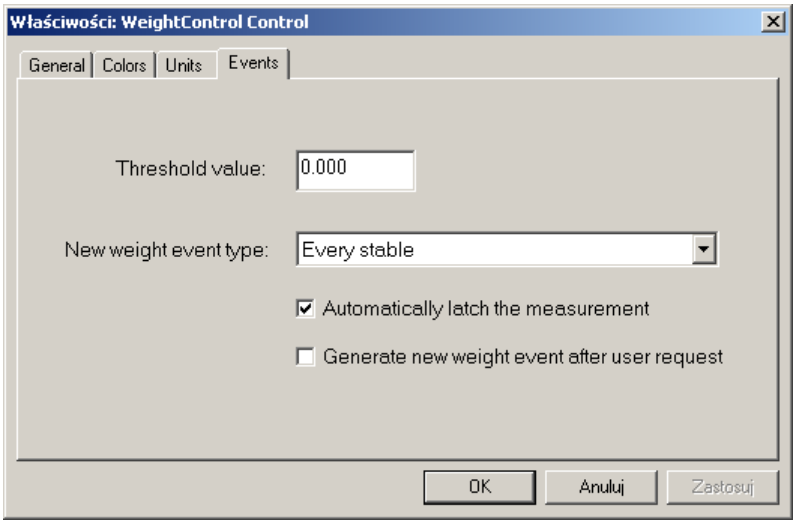
The way the interface looks like can be changed by setting colours. The following elements can have colours changed: Foreground and background colour, display area colour and colours of buttons.

2.3.3 Units



The default unit is the main unit that is used for displaying after the program starts running. The rest units can be ticked as accessible for the UNITS button so that an operator could toggle between them.

2.3.4 Events

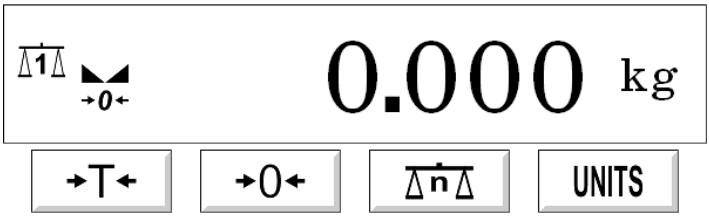


The control can generate events when the predefined circumstances appear.
The event ***new_weight*** informs about a new weighing result. It is configurable by setting properties below:

- **Threshold value** – weighing value over which an event can be generated accordingly to another parameter (event type).
- **New weight event type** – event type, possible settings:
  - **Disabled** – event not generated.
  - **Every stable** – every stable result generates the event, threshold value is ignored.
  - **First stable above the threshold value** – only first stable result over the threshold generates the event, next event can be generated after indications return below the threshold and then again exceeded.
  - **Last stable above the threshold value** – only the last stable result above the threshold generates the event.
- **Automatically latch the measurement** – If generated, it causes latching the measurement for reading. The latched measurement can be accessed by calling the right method (see chapters below).
- **Generate new weight event after user request** – If selected, it causes generating the event only upon the request of user, the event can be requested by calling the right method. This is a single request (one request causes one event).

2.4. Operating

After running the control may look as below:



According to the setting following pictograms are displayed:

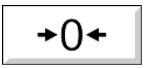
- first platform selected

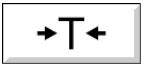
- precise zero indicated

- stable indication

kg - weighing unit

2.4.1 Button's functions

 Zeroing

 Tarring

Toggling between platforms

Toggling between weighing units

2.4.2 Error messages

Err2 - Value beyond the zero range,

Err3 - Value beyond the tare range,

Err8 - Tarring / zeroing operation time exceeded,

NULL - Zero value from the AD converter,

FULL - Measurement range overflow,

LH - Start mass error, the mass on the weighing platform is beyond the acceptable range ±10 of start mass

2.5. Accessible Methods

- Scale zeroing and tarring:

  void Zero(void);
  void Tare(void);

- Toggling between units:

  SHORT ChangeUnit(SHORT newUnitCode);
  newUnitCode – unit code (0-kg, 1-lb, 2-oz, 3-ct, 4-N, 5-g),
  if the selected unit is disabled the closest accessible unit is selected.

- Getting unit in string format:

  BSTR GetUnit(void);

- Getting mass in grams (g):

  DOUBLE GetWeightSi(void);

- Set of functions for reading different features of the scale:

  FLOAT GetWeightFloat(void);
  SHORT GetUnitInt(void);
  FLOAT GetTareFloat(void);
  SHORT GetFractionLenght(void);
  SHORT GetMode(void);
  SHORT GetOperationStatus(void);
  SHORT IsStatusZero(void);
  SHORT IsStatusStable(void);
  SHORT IsStatusTare(void);
  SHORT IsStatusOK(void);
  SHORT IsStatusFULL(void);
  SHORT IsStatusNULL(void);
  SHORT IsStatusLH(void);
  SHORT IsStatusRange2(void);
  SHORT IsStatusRange3(void);
  SHORT IsCorrectWeight(void);

- Latching the copy of indication in the control:

  SHORT Latch_measurement(void);

- Subset of functions for reading of the latched measurement:

  FLOAT LGetWeightFloat(void);
  SHORT LGetUnitInt(void);

FLOAT LGetTareFloat(void);
SHORT LGetFractionLenght(void);
SHORT LGetMode(void);
SHORT LGetOperationStatus(void);
SHORT LIsStatusZero(void);
SHORT LIsStatusStable(void);
SHORT LIsStatusTare(void);
SHORT LIsStatusOK(void);
SHORT LIsStatusFULL(void);
SHORT LIsStatusNULL(void);
SHORT LIsStatusLH(void);
SHORT LIsStatusRange2(void);
SHORT LIsStatusRange3(void);
SHORT LIsCorrectWeight(void);
SHORT GetLongOperationStatus(void);
SHORT LGetLongOperationStatus(void);

- The event request (useful if parameter „**Generate new weight event after user request**" is set):

  void RequestNWEvent(void);

- Set of functions operating in the counting pieces mode

  FLOAT GetPcs(void) *- get number of piecess;*
  DOUBLE GetPcsPattern (void) *– get reference unit mass;*
  SetPcsPattern (DOUBLE*) – set reference unit mass.*

2.6. Generated events

- Configurable event (possible settings in chapter 1.4):

  void new_weight(void);

- Scale status change:

  void status_change(int code);
  code – status codes (-1 – FULL, -2 – NULL, -3 – LH, 0 – OK).

- Status change of long lasting operations (tarring, zeroing):

  void pending_status(int code);
  code – status codes (-5 – time overflow, -6 – tarring beyond the range -7 – zeroing beyond the range, 0 – status ok).

- Stability change

  void stable_change(SHORT stable);
  stable – state of stability(1 – stable, 0 – non-stable).

3. DYNAMIC-LINK LIBARIES (DLL) FOR PUE C/31

3.1. Intended Use

DLL (*Dynamic-Link Library)* is intended for Windows environment. It includes functions' implementations for cooperation with PUE C/31 indicators and compatible scales (WLC and WTB). Those functions can be called in programs made by customers and 3rd party companies that use such programming tools as VB, C++, C#.

3.2. Installation

Package files:

- **PueC31Client.dll**
  This .dll file should be placed in the folder where the executable file of a created program is placed,
- **PueC31Client.lib**
  This file is intended to attach to customers' C++ projects,
- **PueC31Client.h**
  The library header, intended to attach to customers' C++ projects,
- **PueC31Client.cs**
  Libraries implemented in a class, intended to attach to customers' C# projects.

The functions included in the library works as client-server with ***„PueC31 controller"***, *installed on a computer that is intended to cooperate with the scale.*

3.3. Functions

3.3.1 Communication with a scale

In order to communicate with a scale the following parameters should be set:

- **IP** – computer IP address with installed the ***„PueC31 controller"*** *and attached a scale*,
- **timeout** – the maximal time interval for receiving a response from a scale,

- **No_repeat** – the number of failure tries in establishing communication with a module that causes the error message,
- **port_number** – UDP port number for communication with ha scale (see appendix A).

All this settings are put into the ***com_conf_st*** structure using the ***set_com_conf*** function.

Caution:
*It should be remembered that the memory taken by the **com_conf_st** structure, should be released using the **free_com_conf_st** function.*

3.3.1.1 Creating the structure com_conf_st structure comprising communication parameters.

com_conf_st set_com_conf(char* _IP, int _address, int _timeout, int _repeat_number);*

Parameters:

- ***IP*** – computer IP address with connected a scale,
- ***timeout*** - the maximum time interval in milliseconds for receiving a response from an external device,
- ***repeat_number*** - the number tries to establish communication with a device,
- **port_number** – UDP port number for communication with ha scale (see appendix A).

Returned value:

config - Pointer to the structure comprising communication parameters.

3.3.1.2 Function releasing memory occupied by the com_conf_st structure

int free_com_conf_st(com_conf_st config);*

Parameters:

- ***config*** – Pointer to the structure ***com_conf_st*** comprising communication parameters.

Returned value:

1 – memory released

3.3.1.3 Sending request and receiving responses

- *int request_com_conf(com_conf_st* config, const char* query_text, char* response_buffer);*

Parameters:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,
- ***query_text*** – string of characters comprising a request terminated by '\0',
- ***response_buffer*** – buffer to which the function inscribes a response, allocated array of characters.

Returned value – Operation status:

- 0 – operation successful, status OK
-1 – device timeout
-5 – port cannot be opened
-6 – another exception (server)
-7 – communication timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – another error (client)

In case of points -6,-9 – details about an exception/error can be found

- *int request_com_confL(com_conf_st* config, const char* query_text, char* response_buffer, int query_lenght);*

Parameters:

- ***config*** - - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,
- ***query_text*** - string of characters comprising a request,
- ***response_buffer*** - buffer to which the function inscribes a response, allocated array of characters,
- ***query_lenght*** – length of request.

Returned value – Operation status:

0 – operation successful, status OK
-1 – device timeout
-5 - port cannot be opened
-6 – another exception (server)
-7 – communication timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – another error (client)

In case of points -6,-9 – details about an exception/error can be found

- *int request(const char* IP, const char* query_text, int timeout, char* response_buffer, int repeat_number , int portnumber);*

Parameters:

- ***IP*** – computer IP address with connected a scale,
- ***timeout*** - the maximum time interval in milliseconds for receiving a response from an external device,
- ***query_text*** - string of characters comprising a request terminated by '\0',
- ***response_buffer*** - buffer to which the function inscribes a response, allocated array of characters,
- ***repeat_number*** - the number tries to establish communication with a device,
- **portnumber** – UDP port number for communication with ha scale (see appendix A).

Returned value – Operation status:

0 – operation successful, status OK
-1 – device timeout
-5 – port cannot be opened
-6 – another exception (server)
-7 – communication timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – another error (client)

In case of points -6,-9 – details about an exception/error can be found.

- *int requestL(const char* IP, const char* query_text, int timeout, char* response_buffer, int repeat_number, int query_lenght, int portnumber);*

Parameters:

- ***IP*** – computer IP address with connected a scale,
- ***timeout*** - the maximum time interval in milliseconds for receiving a response from an external device,
- ***query_text*** - string of characters comprising a request terminated by '\0',
- ***response_buffer*** - buffer to which the function inscribes a response, allocated array of characters,
- ***repeat_number*** - the number tries to establish communication with a device.
- ***query_lenght*** – length of request,
- **portnumber** – UDP port number for communication with ha scale (see appendix A)

Returned value – Operation status:

0 – operation successful, status OK
-1 – device timeout
-5 – port cannot be opened
-6 – another exception (server)
-7 – communication timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – another error (client)

In case of points -6,-9 – details about an exception/error can be found.

3.3.2 Operations on structure measurement_st

A complete measurements consists of:

- Mass value,
- Tare value,
- Unit code,
- Scale state.

All these data are included in structure ***measurement_st*** and can be read by one command - ***get_measurement_st***. Structure components can be separated by using appropriate functions.

Caution:

It should be remembered that the memory taken by the ***measurement_st*** *structure, should be released using the* ***free_measurement_st*** *function.*

3.3.2.1 Reading measurement from a scale

measurement_st get_measurement_st(com_conf_st* config);*

Parameters:

- ***config*** - Wskaźnik do struktury zawierającej konfigurację parametrów komunikacji, zainicjalizowanej przez funkcję set_com_conf_st.

Returned value:

measurement – Pointer to structure ***meas*** comprising a measurements.

3.3.2.2 Releasing memory occupied by the measurement_st structure

int free_measurement_st(measurement_st measurement);*

Parameters:

- ***measurement*** – Pointer to structure ***meas*** comprising a measurement.

Returned value:

1 – memory released

3.3.2.3 Reading mass value in floating point format

double get_weight_float(measurement_st measurement);*

Parameters:

- ***measurement*** - Pointer to structure ***meas*** comprising a measurement.

Returned value:

weight – mass reading.

3.3.2.4 Reading mass value in string format

int get_weight_string(measurement_st measurement, char* weight);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement,
- ***weight*** – pointer to the allocated array of characters for mass value.

Returned value:

Operation status

3.3.2.5 Checking stability of the scale

int is_status_stable(measurement_st measurement);*

Parameters

- **measurement** - pointer to the structure ***meas*** holding the measurement.

Returned value – stability status:

1 - stable
0 - unstable

3.3.3 Reading weighing unit in string format

int get_unit_string(measurement_st measurement, char* unit);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.
- ***unit*** – pointer to the allocated array of characters for unit value.

Returned value:

Reading operation status

3.3.3.1 Unit reading in the fixed-point format

int get_unit_int(measurement_st* measurement);

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

Measuring unit.

```
enum units
{
    kg = 0,
    g = 1,
    N = 2,
    lb = 3,
    ct = 4,
    pcs = 5,
    oz = 6,
    pct = 7
};
```

3.3.3.2 Reading tare value in floating point format

double get_tare_float(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

Tare value.

3.3.3.3 Reading tare value in string format

int get_tare_string(measurement_st measurement, char* tare);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement,
- ***tare*** – pointer to the allocated array for tare value.

Returned value:

Reading operation status.

3.3.3.4 Checking the measurement correctness

int is_correct_weight(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement,

Returned value:

1 – measurement correct
0 - measurement incorrect

3.3.3.5 Checking if the scale has been zeroed

int is_status_zero(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value - zero status:

1 – zeroed
0 – out of zero range

3.3.3.6 Checking if the scale has been tarred

int is_status_tare(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value – tare status:

1 – Tare > 0
0 – Tare = 0

3.3.3.7 Checking if the range is exceeded

int is_status_FULL(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value - FULL status:

1 – over the range
0 – within the range

3.3.3.8 Checking if a measurement is below the range

int is_status_NULL(measurement_st measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value - NULL status:

1 – below the range
0 – within the range

3.3.3.9 +/- control in relation to a set standard

- *int is_range_OK(measurement_st* measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value:

1 – mass in a defined range,
0 – mass beyond a defined range

- *int is_range_LO(measurement_st* measurement);*

Parameters:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value :

1 – mass below a defined range
0 – mass not below a defined range

- *int is_range_HI(measurement_st* measurement);*

Parameter:

- ***measurement*** - pointer to the structure ***meas*** holding the measurement.

Returned value :

1 – mass over a defined range
0 – mass not over a defined range

3.3.4 Tarring and zeroing

3.3.4.1 Setting tare value

int set_tare(com_conf_st config, double _tare);*

Parameters:

- ***config*** – pointer to the structure ***com_conf_st*** comprising communication parameters,
- ***tare*** – tare value to be set.

Returned value – operation status :

0 - Done, status OK
-1 – Device Timeout
-5 - Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer response_buffer not allocated
-9 – other error (client)

3.3.4.2 Tare value reading

double get_tare(com_conf_st config);*

Parameters:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function.

Returned value: Tare value

3.3.4.3 Scale tarring

int tarring(com_conf_st config);*

Parameter:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-5 - Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)

3.3.4.4 Scale zeroing

int zeroing(com_conf_st config);*

Parameter:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function.

Returned value – operation status:

0 - Done, status OK
-1 – Device Timeout
-5 - Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)

3.3.5 Operations on parameters

Parameters in PUE C/31 indicators and compatible derivatives are saved in three different memories: Flash, RAM and EEPROM.

3.3.5.1 Reading parameters

There are 4 functions for reading parameters. Use them according to a parameter type:

- *char read_param_char(com_conf_st* config, WORD address, mem_type_en mt);*
- *int read_param_int16(com_conf_st* config, WORD address, mem_type_en mt);*
- *int read_param_int32(com_conf_st* config, WORD address, mem_type_en mt);*
- *float read_param_float(com_conf_st* config, WORD address, mem_type_en mt);*

Parameters:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,
- ***address*** – Parameter address.
- ***mt*** – memory type:

```
enum mem_type_en
{
F=1, //flash
R=2, //ram
E=3 //eeprom
};
```

Returned value – parameter value**.**

3.3.5.2 Writing parameters

There are 4 functions for writing parameters. Use them according to a parameter type:

- *int write_param_float(com_conf_st* config, WORD address, mem_type_en mt, float value);*
- *int write_param_int32(com_conf_st* config, WORD address, mem_type_en mt, int value);*
- *int write_param_int16(com_conf_st* config, WORD address, mem_type_en mt, int value);*
- *int write_param_char(com_conf_st* config, WORD address, mem_type_en mt, char value);*

Parameters:

- ***config*** - pointer to a structure comprising communication parameters initiated by the ***set_com_conf_st*** function,
- ***address*** – parameter address.
- ***mt*** – memory type:

```
enum mem_type_en
{
F=1, //flash
R=2, //ram
E=3 //eeprom
};
```

- ***value*** – parameter value:

Returned value – operation status:

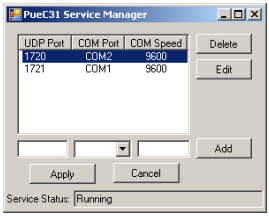
0 - Done, status OK
-1 – Device Timeout
-4 – Operation not confirmed by an external device
-5 - Port cannot be opened
-6 – another thread (server)
-7 – communication Timeout
-8 - Buffer ***response_buffer*** not allocated
-9 – other error (client)

In case of points -6,-9 – details about an exception/error can be found.

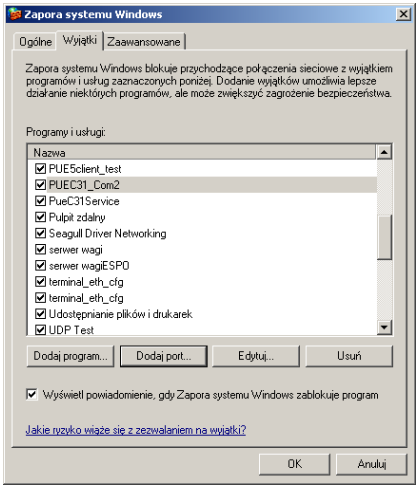
3.4. PueC31 controller configuration

„PueC31 controller” collects data from serial ports RS232, where scales are connected, and sends and sends them to the Ethernet network using appropriate UDP ports. To allow communication with scales the controller should be appropriately configured.

The configuration can be performed using „PueC31 Service Manager” software. It is accessible in the folder where ***„PueC31 controller”*** is installed:

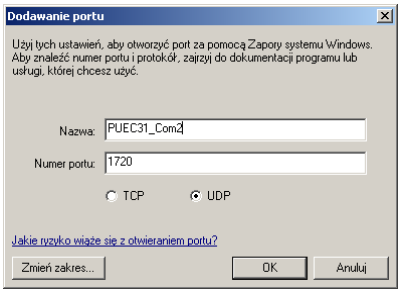


Configuration consist in ascribing a serial port (RS232) to the UDP port and setting a baud rate the same as in the indicator. You need to press the **Apply** button to save settings.



It should be remembered that if we want to communicate with other computers in the net that an exception needs to be added to the
Windows firewall. In Windows XP it can be made as follows:

1. Open the firewall configuration Window, **Exceptions** overlay:
2. Press **Add port** and inscribe UDP ports ascribed for communication with the scale:



3. Confirm changes.

4. ACTIVEX CONTROL FOR INDICATORS PUE C/31

4.1. Intended Use

ActiveX control for **PUE C/31** is intended for using in programs made by customers and 3rd party companies that use such programming tools as VB, C++, C#. ActiveX technology allows to convey data between different applications running under control of Windows operating system. This technology saves time that is necessary to introduce new projects and solves viable communication problems.

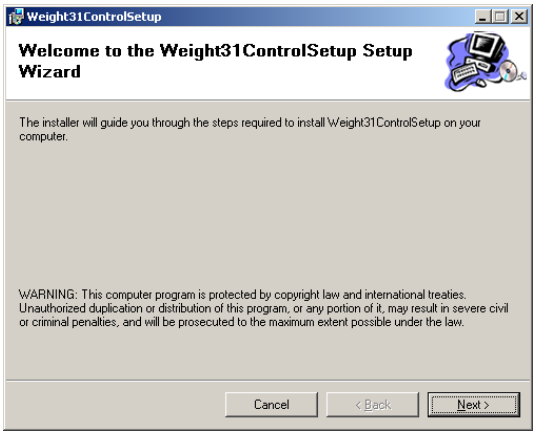
In case of scales with PUE C/31 indicators and compatible (WLC, WTB) the introduced control is a configurable GUI (Graphic User Interface) and allows easy implementation of own PC scale software without knowing communication protocols by setting properties using predefined event handlers and suggested graphics.

Caution:
*This control works as **client-server** with **„PueC31 controller”** installed on a computer that is intended to cooperate with the scale. More information about **„PueC31 controller”** you can find in appendix A.*

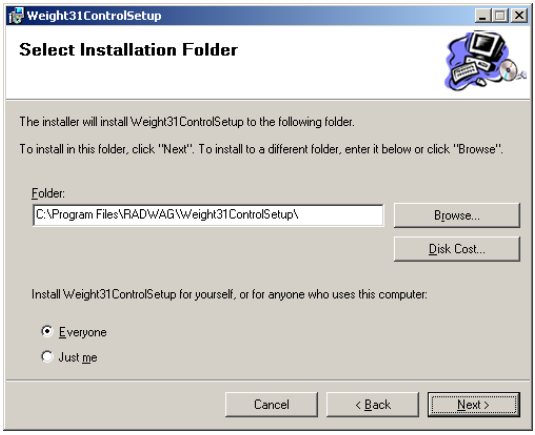
4.2. Installation

Installation procedure for ActiveX control:

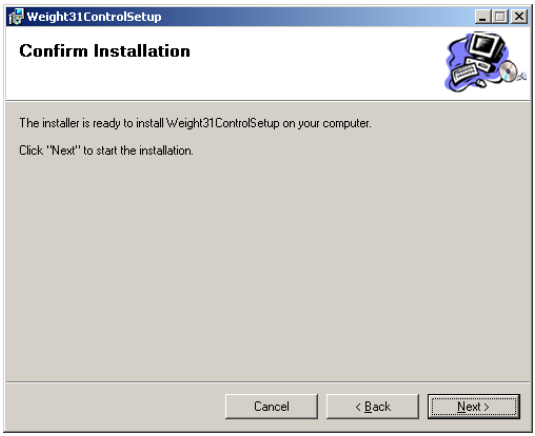
- Run: **setup.exe**, the following window is displayed:


Weight31ControlSetup
Welcome to the Weight31ControlSetup Setup Wizard
The installer will guide you through the steps required to install Weight31ControlSetup on your computer.
WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.
Cancel
< Back
Next >

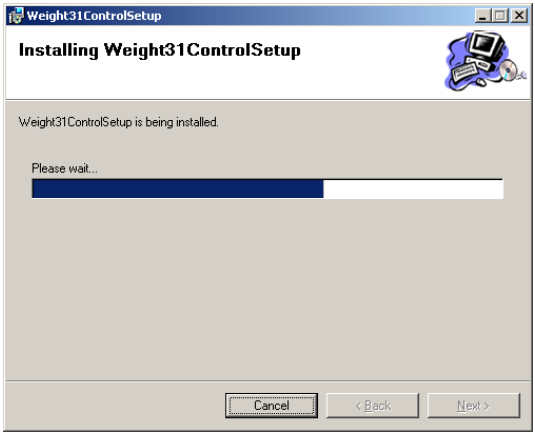
- Go to the next step by pressing Next >,
- Program **Setup.exe** allows the user to chose the folder name for the program:


Weight31ControlSetup
Select Installation Folder
The installer will install Weight31ControlSetup to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files\RADWAG\Weight31ControlSetup\
Browse...
Disk Cost...
Install Weight31ControlSetup for yourself, or for anyone who uses this computer:
Everyone
Just me
Cancel
< Back
Next >

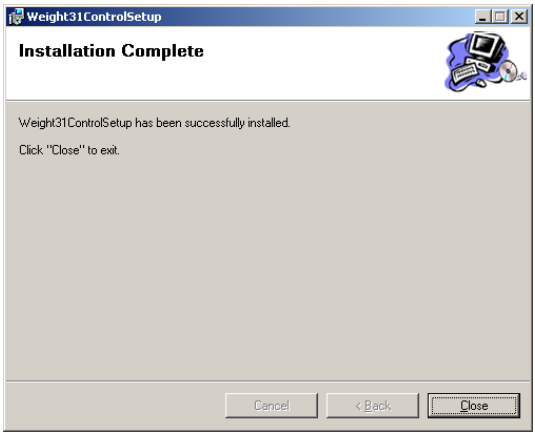
- Go to the next step by pressing Next >,
- Program **Setup.exe** informs the user that it is ready to be installed:


Weight31ControlSetup
Confirm Installation
The installer is ready to install Weight31ControlSetup on your computer.
Click "Next" to start the installation.
Cancel
< Back
Next >

- Go to the next step by pressing Next > and program **Setup.exe** begins the installation process:



- When the installation is completed the following window is displayed:



- Press Close and the installation process is completed.

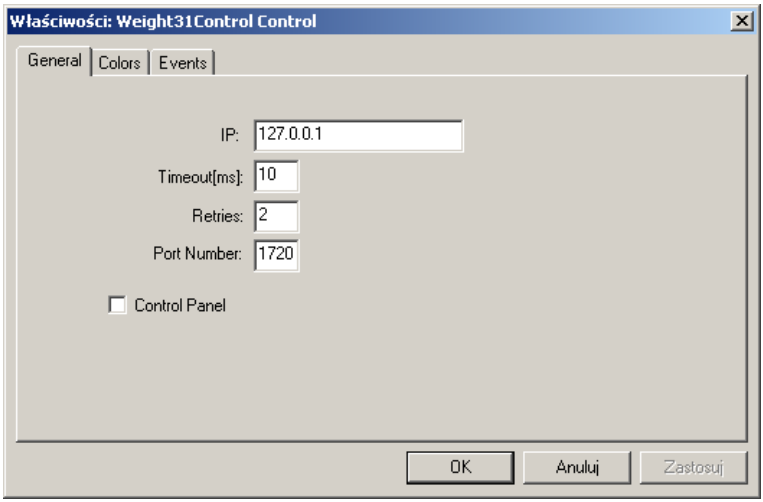
Caution:

1. *The installation of **„PueC31 controller”**, as well as the installation of ActiveX control, is simple, intuitive and do not require additional remarks. If there are no special requirements just accept the default way of installation.*
2. *For appropriate operation of **ActiveX** control it is necessary to install the **„PueC31 controller”**. See appendix A for more details about the installation.*

4.3. Configuration

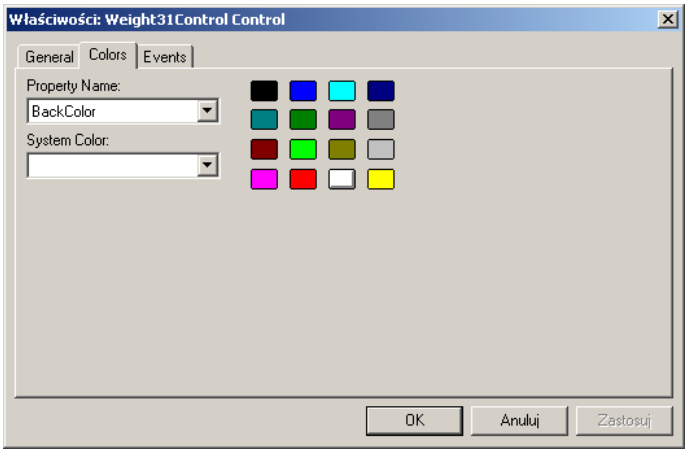
The look and operation of the control are widely parameterized. Default values allow to run the control on PC with ***„PueC31 controller”*** installed. Parameters are divided into three subsets.
“General” parameters include setting of internal communication with the weighing module and enabling/disabling the appearance of buttons.
“Colours” enables you to change the look of the control.
“Events” outlines the conditions that generate program different events.

4.3.1 General parameters



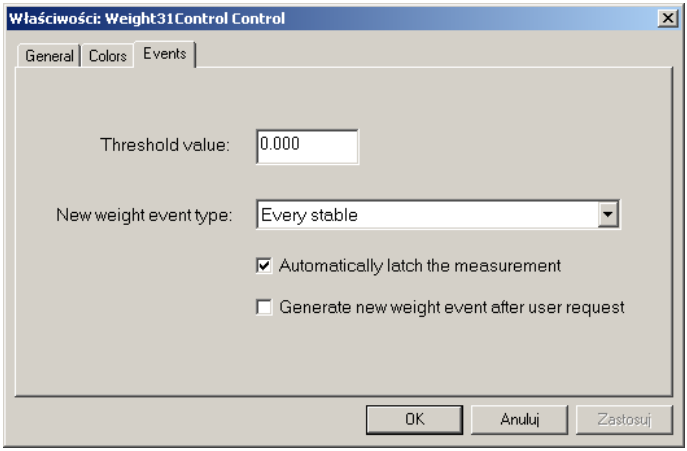
- IP – Internet Protocol Address of weighing terminal,
- Device Address – Weighing module address (set by the slot the module is in),
- Retries – Number of failure module readings that generate an error,
- Control Panel – if ticked buttons are visible,
- Port Number – UDP port number to communicate with an indicator,
- EnablePcs - If set (1), there is a number of pieces displayed in the weight indicator window.

4.3.2 Colors



The way the interface looks like can be changed by setting colours. The following elements can have colours changed: Foreground and background colour, display area colour and colours of buttons.

4.3.3 Events



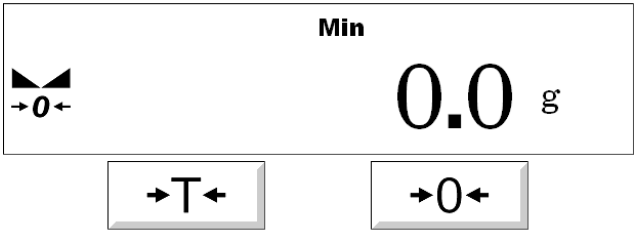
The control can generate events when the predefined circumstances appear. The event ***new_weight*** informs about a new weighing result. It is configurable by setting properties below:

- **Threshold value** – weighing value over which an event can be generated accordingly to another parameter (event type).
- **New weight event type** – event type, possible settings:
  - **Disabled** – event not generated.
  - **Every stable** – every stable result generates the event, threshold value is ignored.
  - **First stable above the threshold value** – only first stable result over the threshold generates the event, next event can be generated after indications return below the threshold and then again exceeded.
  - **Last stable above the threshold value** – only the last stable result above the threshold generates the event.
- **Automatically latch the measurement** – If generated, it causes latching the measurement for reading. The latched measurement can be accessed by calling the right method

- **Generate new weight event after user request** – If selected, it causes generating the event only upon the request of user, the event can be requested by calling the right method (see chapter 4 for details). This is a single request (one request causes one event).

4.4. Operating

After running the control may look as below:



According to the setting and scale state following pictograms are displayed:

→0←	- precise zero indicated
	- stable indication
Net	- set tare
g	- weight unit
Min	- Measurement below an acceptable scope
OK	- Measurement within an acceptable scope
Max	- Measurement over an acceptable scope

4.4.1 Functions of buttons

→0←	Zeroing
→T←	Tarring

4.4.2 Error messages

NULL - Zero value from the AD converter

FULL - Measurement range overflow

4.5. Accessible methods

- Scale zeroing and tarring:

  void Zero(void);
  void Tare(void);
  SHORT SetTare(DOUBLE tare);

- Getting unit in string format:

  BSTR GetUnit(void);

- Getting mass in grams (g):

  DOUBLE GetWeightSi(void);

- Set of functions for reading different features of the scale:

  FLOAT GetWeightFloat(void);
  SHORT GetUnitInt(void);
  FLOAT GetTareFloat(void);
  SHORT IsStatusZero(void);
  SHORT IsStatusStable(void);
  SHORT IsStatusTare(void);
  SHORT IsStatusFULL(void);
  SHORT IsStatusNULL(void);
  SHORT IsCorrectWeight(void);

- Latching the copy of indication in the control:

  SHORT Latch_measurement(void);

- Subset of functions for reading a latched measurement:

  FLOAT LGetWeightFloat(void);
  SHORT LGetUnitInt(void);
  FLOAT LGetTareFloat(void);

SHORT LIsStatusZero(void);
SHORT LIsStatusStable(void);
SHORT LIsStatusTare(void);
SHORT LIsStatusFULL(void);
SHORT LIsStatusNULL(void);
SHORT LIsCorrectWeight(void);

- The event request (useful when parameter „**Generate new weight event after user request**" is set):

  void RequestNWEvent(void);

- Set of functions operating in the counting pieces mode

  FLOAT GetPcs(void) *- get number of piecess;*
  DOUBLE GetPcsPattern (void) *– get reference unit mass;*
  SetPcsPattern (DOUBLE) *– set reference unit mass.*

4.6. Generated events

- Configurable event (possible settings in chapter 1.4):

  void new_weight(void);

- Scale status change:

  void status_change(int code);
  code – status codes (-1 – FULL, -2 – NULL, 0 – OK).

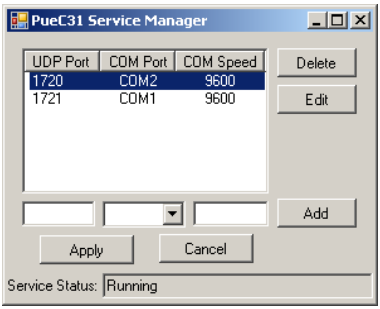
- Stability change

  void stable_change(SHORT stable);
  stable – state of stability(1 – stable, 0 – non-stable).

4.7. PueC31 controller configuration

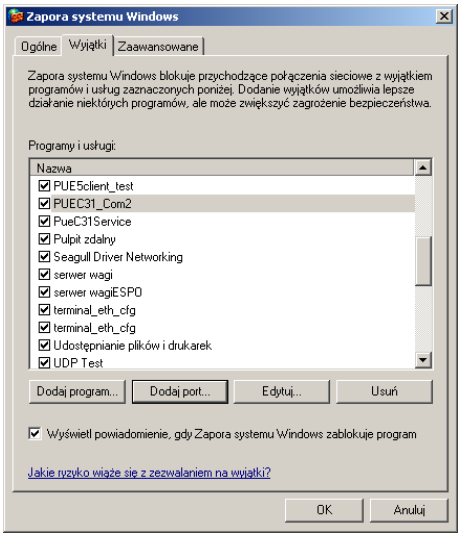
„PueC31 controller" collects data from serial ports RS232, where scales are connected, and sends and sends them to the Ethernet network using appropriate UDP ports. To allow communication with scales the controller should be appropriately configured.

The configuration can be performed using „PueC31 Service Manager" software. It is accessible in the folder where ***„PueC31 controller"*** is installed:



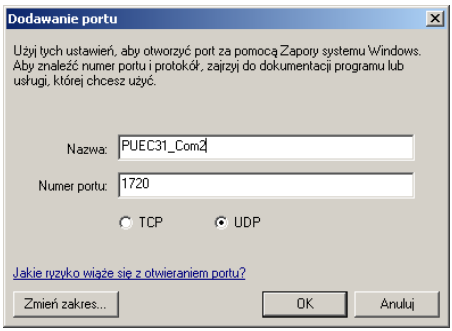
Configuration consist in ascribing a serial port (RS232) to the UDP port and setting a baud rate the same as in the indicator. You need to press the **Apply** button to save settings.

It should be remembered that if we want to communicate with other computers in the net that an exception needs to be added to the
Windows firewall. In Windows XP it can be made as follows:



4. Open the firewall configuration Window, **Exceptions** overlay:

5. Press **Add port** and inscribe UDP ports ascribed for communication with the scale:



6. Confirm changes.

5. PROFIBUS COMMUNICATION MODULE

5.1. Intended use

AnyBus CompactCom – compact communication port including a standard RISC processor. In the **Profibus-DPV1-AB6200** version there is accessible 244 bytes a cyclic buffer for data transmitted in both directions and additionally some acyclic parameters. It is possible to communicate with master stations both class 1 and class 2. The connector is galvanically insulated and allows automatic baud rate detection within the range of 9.6 kbit/s to 12 Mbit/s. There is a gsd file delivered together with the module.

5.2. Data exchange

NOTICE!
The information on data exchange with the weighing module has been presented in this chapter. The information on the PROFIBUS configuration can be found further in the document.

Inputs:

variable	address	Word length	Data type
Command	0	1	byte
Tare	1	2	float

- command:
  - zero – 1(bit 0)
  - tare – 2(bit 1)
  - set tare – 4(bit 2), (variable holding a tare value)
- tare – given tare value,

Outputs:

variable	address	Word length	Data type
Mass	0	2	float
status	4	1	word
tare	6	2	float

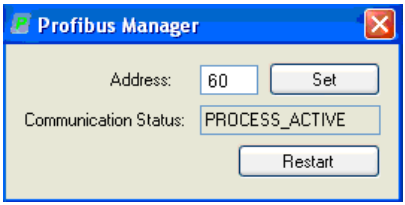
- mass – indication (measurement),
- status bits:
  - 0 – measurement OK (no errors announced)
  - 1 – stable measurement
  - 2 – zero indication
  - 3 – tare memorized
  - 4 – indication in range two
  - 5 – indication in range three
  - 6 – error announced - NULL
  - 7 – error announced - LH
  - 8 – error announced - FULL
- tare – set tare value.

5.3. Diagnostics and operations on the ProfiBus module

There is an icon in the taskbar showing communication status. It can be found in the right bottom corner of the screen:

- communication on
- communication off
- communication suspended
- communication error

In order to open the ProfiBus Manager click on the icon using the right mouse button and select „open”:



- Setting address – to attribute a new address to the ProfiBus station type it in the field „Address” press „Set” and restart communication.

5.4. Memory map

Output:

Address	0	1	2	3	4	5	6	7	8	9
0	M	M	M	M	S	S	T	T	T	T
1										

Input:

Address	0	1	2	3	4	5	6	7	8	9
0	K	T	T	T	T					
1										

M – Mass, 4 bytes, float, little endian
S – Status, 2 bytes
T – Tare, 4 bytes, float, little endian
K – command byte

5.5. Configuration

In order to make the Profibus operate appropriately the USB Serial port needs to be set to COM4 in the Windows Device Manager. Error **128** is removed by restart. The default address is 60.

5.6. Cable for module PROFIBUS

In order to describe connecting the PROFIBUS module via a gland in the back wall of the housing there is a cable led through the gland:

1A	Green
1B	Yellow or Red

5.7. All fields used in Profibus communication

Output data from the weighing module

"Weight"	FLOAT	[4 byte]	
"Status"	UINT16	[2 byte]	
"Tare"	FLOAT	[4 byte]	
"ID"	UINT32	[4 byte]	Not used in the basic version
"TV"	FLOAT	[4 byte]	Not used in the basic version
"SUM"	FLOAT	[4 byte]	Not used in the basic version
"Signal"	BOOL	[byte]	Not used in the basic version

Input data for the weighing module

"Command",	BOOL	[byte]	
"RefTare"	FLOAT	[4 byte]	
"RefID"	UINT32	[4 byte]	Not used in the basic version
"RefTV"	FLOAT	[4 byte]	Not used in the basic version
"RefThre"	FLOAT	[4 byte]	Not used in the basic version
"P"	UINT32	[4 byte]	Not used in the basic version

5.8. PROFIBUS configuration example

In this example application the MASTERA station is „CIF 50-PB" PCI card. The full configuration printout is included in file **Profibus print configuration .pdf,** that is recorded on the CD.

Slave60

Stadionaddress: 60
Device: Anybus-CC PROFIBUS DP-V1
IDENT Number: 0x1811
GDS File: HMS_1811.GDS
GDS-Revision:

PROFIBUS-DP Address Table

Module Name	Slot	Idx.	Input type	Input Offset	Output Type	Output Offset
Output 1 byte	1	1			Byte	0
Output 2 words	2	1			Word	1
Output 2 words	3	1			Word	5
Output 2 words	4	1			Word	9
Output 2 words	5	1			Word	13
Output 2 words	6	1			Word	17
Input 2 words	7	1	Word	0		
Input 1 word	8	1	Word	4		
Input 2 words	9	1	Word	6		
Input 2 words	10	1	Word	10		
Input 2 words	11	1	Word	14		
Input 2 words	12	1	Word	18		
Input 1 byte	13	1	Byte	22		

Parameter Data

Byte	Description	Value
0	1 parameter data byte	0x80
1	2 parameter data byte	0x00
2	3 parameter data byte	0x00

5.9. Content of CD

A CD with source codes of programs are attached to PUE5 terminals equipped with the PROFIBUS AB6200 module. It allows customers and 3rd party companies to adjust terminals equipped with AB6200 modules to specific customers needs.

CIF32TEST	-	Source code of the simple program (C#) using cif32dll.dll library to copy data from the CIF50DP PCI card (master profibus),
DPV1Slave	-	Source code of driver library to supervise the communication module AB6200 (slave profibus). To change the quantity of data sent in transmission some changes needs to be done in the library code and it needs to be recompiled. In module AB6200 there are 244 bytes of data at disposal for input data and 244 bytes of data at disposal for output data. Basic source code released by the producer of the communication module. Module configuration in files appd.h and appd.c.
DPV1Share	-	The library that gives access to the shared memory space for data exchange (memory space structured in the same way as in DPV1Slave),
ProfibusManager	-	Source code of the application controlling operation of AB6200 module and weighing module,
PUE5_OCX_PROF	-	Source code of weighing module client "Scale display".

Current documentation concerning AB6200 modules can be found on the producer website – HMS company
http://www.anybus.com/support/support.asp?PID=321&ProductType=Anybus-CompactCom

6. COMPONENT OF COMMUNICATION WITH BAR-CODE SCANNER

6.1. Intended use

This software component is used to read barcodes from a scanner via the RS232 serial interface. It can be used in programming environments and languages supporting the .NET technology.

6.2. Installation

Package files:

- **scan.dll**
  File .dll needs to be attached to the project,

In the Visual Studio environment the best way to get access to the component is to add it to the list of controls (toolbox), then it can be placed on any form and the library is automatically attached to the project.

6.3. Properties

6.3.1 Form - Ctrl

Select a form which receives data from the scanner component. It is an object of System.Windows.Forms.Control type, in which we want to read barcodes.

6.3.2 Serial port number – PortNumber

Select a serial port to which a barcode scanner is connected – possible selection: COM1 to COM10, default COM1.

6.3.3 Baud rate – PortSpeed

Select baud rate to communicate with a scanner – default 9600 bps.

6.3.4 Telegram prefix length – PrefixNumber

Select the number of characters at the beginning of the telegram to skip during transmission – default 0 characters.

6.3.5 Telegram suffix length – PostfixNumber

Select the number of characters in the end of the telegram to skip during transmission – default 0 characters.

6.4. Methods

6.4.1 Starting reception telegrams from a scanner

Start() – the method opens the selected serial port and starts “listening” to telegrams sent with the specified baud rate.

6.4.2 Completion of reception of telegrams from a scanner

Stop() – the method ends “listening” to telegrams and closes the selected serial port.

6.5. Events

6.5.1 Reading a new code

NewCode – the event is generated when a scanner has read a code. In a function parameter there is a code placed in string format.

MANUFACTURER

OF ELECTRONIC WEIGHING INSTRUMENTS

RADWAG WAGI ELEKTRONICZNE
26 – 600 Radom, Bracka 28 Street
POLAND
Tel. +48 48 38 48 800, tel./fax. + 48 48 385 00 10
Selling departament + 48 48 366 80 06
www.radwag.com

DIN EN ISO 9001:2000
CERTIFICATE NO 71 100 C206